APPENDIX B

9(a)----------

see the source code under headings 1(a), 1(b) and 1(c) in Appendix A

APPENDIX B

9(b)----------

note: In the JobDB class, GetJob is the method that permits a client to obtain a job from the database. SearchSet is the method that performs the search in the database and makes the Job object. The communication with the job management apparatus is through this Job object as in the Submit method included above. [relevant to Claims 9, 17, 24]

source code excerpt:

```
    # GetJob(id: basics.JobID): job.Job
    def GetJob (self, id):
        try:
            flog.Log("jdb", 'GetJob with id %s' % str(id))
            all = self.SearchSet ([StrAttribute ('ID', id)])
            if len(all) == 0:
                raise basics.DoesNotExist, id
            elif len(all) > 1:
                print "Error: multiple records with id %s." % id
                return all[0]
            else:
                return all[0]
        except basics.DoesNotExist, msg: raise basics.DoesNotExist, msg
        except basics.InvalidID, msg: raise basics.InvalidID, msg
        except:
            # Pass debugging information through for all other errors
            error_str = strfile.StrFile()
            traceback.print_exc(file=error_str)
            error_msg = error_str.read(error_str.size())
            raise basics.Error, error_msg


    # returns a list of Jobs that match the specified list of attributes
    # there is no notion of 'indices' in the JobDB
    def SearchSet(self, attrs):
        list = []
        # Try using the superclass implementation to take advantage of
indexing
        # Then turn each element into a Job rather than a plain Attributed
        result = PersistentCollection.SearchSet(self, attrs)
        for elt in result:
            # Double check that we don't have a sub-attributed
            if elt.parent == 'R':
                list.append(self.make_job(self.elt_index(elt.prefix),0))
        return list

        # NOTE -- Following code is intentionally not reached.
        candidates = self.ilist
        for i in candidates:
            # We only want to return Jobs here, not subattributeds of Jobs
            # Note: with switch to in-memory ilist we're also keeping
            # only roots in the ilist, but the check here is preserved
            # for safety should things change again.
            if not self.rootp_for_elt(i):
                continue
            elt = self.make_job(i, 0)
            try:
```

```
            if elt.Matchp(attrs): list.append(elt)
        except attributed.AttributeUnknown:
            pass
    return list
```

APPENDIX B

9(c) ----------

notes: In the Provider class, AskForWork is the implementation of that part of a provider that communicates with the job management apparatus to receive an assigned task (via the call to AssignWork). Perform is the implementation of that part of a provider that executes the task-specific function to perform the task. SubmitReport is the implementation of that part of a provider that returns a result (via the call to AcceptReport). The Provider class is a superclass so that a plurality of different providers capable of performing different tasks all inherit and use the functionality of the Provider class. The four single-line class definitions are taken from four of our implemented service providers, all inheriting the functionality of the AskForWork, Perform, and SubmitReport methods of the Provider superclass. [relevant to Claims 9, 17]

source code excerpt:

```
    def AskForWork(self):                   # return true when done
        if self.Ready():
            log.log('%s: requesting work from %s', (self.me,
rpr(self.assigner)), sev=log.LOG_DEBUG)
            asmt = self.assigner.AssignWork(self.id, self.manager, 10)
            if asmt.op != pcontact.Operation.Standby:
                log.log('%s: received assignment %s', (self.me, asmt.id))
        else:
            log.log('%s: not ready for work, idling', (self.me,))
            asmt = None

        self.NotifyAssignment(asmt)
        if not asmt or asmt.op == pcontact.Operation.Standby: return
self.Idle()
        else: self.Perform(asmt)

    def Perform(self, asmt):
        self.idlecount = 0
        opname = pcontact.Operation.__image__[asmt.op]
        log.log('%s: Performing %s', (self.me, opname))
        try:
            self.asmt = asmt
            report = getattr(self, opname)(asmt)
            if not report: report = self.DefaultReport(opname)
        except status.Report, report:
            log.log("raised report %s", (str(report),), sev=log.LOG_DEBUG)
            pass
        except:
            self.asmt = None
            log.log('%s: error during assignment %s', (self.me, asmt.id),
sev=log.LOG_ERR)
            msg = self.Message(msgtype=status.MessageType.SystemError,
                               msgargs=['error during assignment',
                                        '%s in %s() method: %s' %
(sys.exc_type, opname, sys.exc_value)])
            report = self.FailureReport(messages=[msg])
            # this is moved down here, since we've seen the behavior of an
exception
            # being thrown inside this call. effectively stomping over the
original
```

```
            # exception - Harry 6/25/98
            callstack.writeexc()
        self.asmt = None
        self.HandleReport(asmt, report)

    def SubmitReport(self, asmt, report, done=1):
        if asmt and asmt.contact:
            log.log('%s: %s for %s (%s,%s) to %s', (self.me,
                ['submitting update', 'reporting completion'][done],
asmt.id,
                status.State.__image__[report.state],
                status.SecondaryState.__image__[report.secondarystate],
                rpr(asmt.contact)), sev=log.LOG_DEBUG)
            return asmt.contact.AcceptReport(self.id, asmt.id, done, report)
        else:
            return pcontact.ReportResponse.Accepted


class g4tif2ps(provider.Provider):
class Distiller(provider.Provider):
class WebGIF(provider.Provider):
class OcrProvider(provider.Provider):
```

APPENDIX B

9(d) ----------

note: In the ProviderManager class, the ReportNow method is the implementation of part of a provider manager that involves communication with service providers (via the call to ReportNow on a service provider) and monitors the tasks being performed (here detecting when a provider has crashed while performing a task). As with the provider code above, the ProviderManager class is a superclass, and the four single-line definitions come from the implementations of four provider managers.
[relevant to Claim 9]

source code excerpt:

```
    def ReportNow(self, provname, asmtid, contact):
      # If the Provider terminates we will reap it at some point
      # Thus if we still have a record of the Provider, assume it is
      # still working and send it a signal to request a report
      if self.namepid.has_key(provname):
          # original approach was to use a signal, but that
          # is fundamentally flawed because it could interrupt
          # any operation in progress with a completely unexpected
          # IOError from which recovery is not possible in general.
          handle = self.GetProvPrivate(provname, self.namepid[provname])
          if handle:
            # For maximum flexibility we're going to interpret
            # various failures as indicating that the Provider is
            # not able to support the communication, in which case
            # there's nothing we can do to provide a reasonable
            # report.
            try:
                handle.ReportNow()
                return
            except (ilu.IluGeneralError, ilu.IluProtocolError,
                    ilu.IluUnimplementedMethodError):
                return
                return

      # Provider dead and obviously didn't report to its contact
      # so file a crash report for it here
      report = status.Report(status.State.Done, status.SecondaryState.Crash,
0, [])
      try:
          contact.AcceptReport(provname, asmtid, 1, report)
      except basics.Error:
          # Problem is likely that the assignment was reported
          # complete by the Provider before we had a chance to
          # get in our report, so ignore error
          pass


class g4tif2psManager(pmanager_impl.ProviderManager): pass
class DistillerManager(pmanager_impl.ProviderManager): pass
class WebGIFManager(pmanager_impl.ProviderManager): pass
class OcrManager(pmanager_impl.ProviderManager): pass
```